TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

ORNL Vampir/VampirTrace Training

Performance Analysis for Hardware Accelerators

Jens Doleschal (jens.doleschal@tu-dresden.de)

Guido Juckeland (guido.juckeland@tu-dresden.de)

ZIH
Center for Information Services & High Performance Computing

Agenda

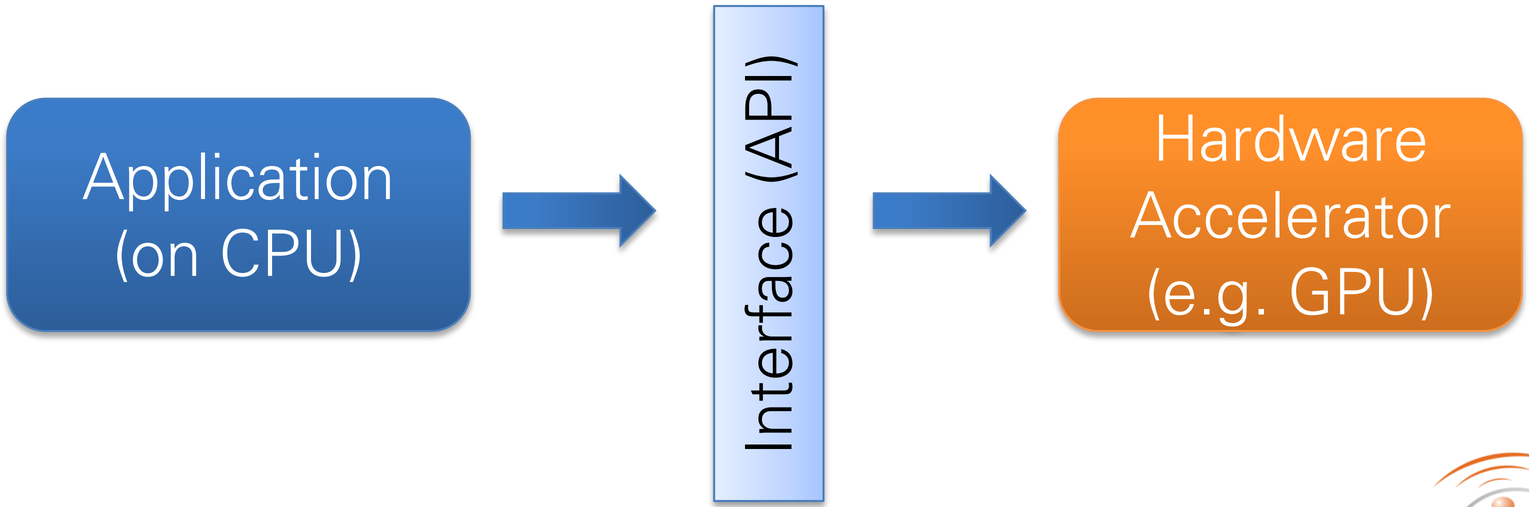
- Motivation / Vendor Support
- VampirTrace API Tracing
- VampirTrace CUDA Support
- An example: PiConGPU
- Summary / Outlook

Motivation / Vendor Support

Motivation

- Single CPU performance stagnating since 2004
- Solution of the CPU vendors → multiple CPU cores on one chip
- BUT: This requires parallelization of the applications to make use of all cores
- (Re-) Appearance of Hardware Accelerators to offer even more performance

Running an accelerated application:



TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

Potential Performance Problems

Accelerator code is (automatically) generated

Accelerator is running its own program code

Not enough insight into the program execution

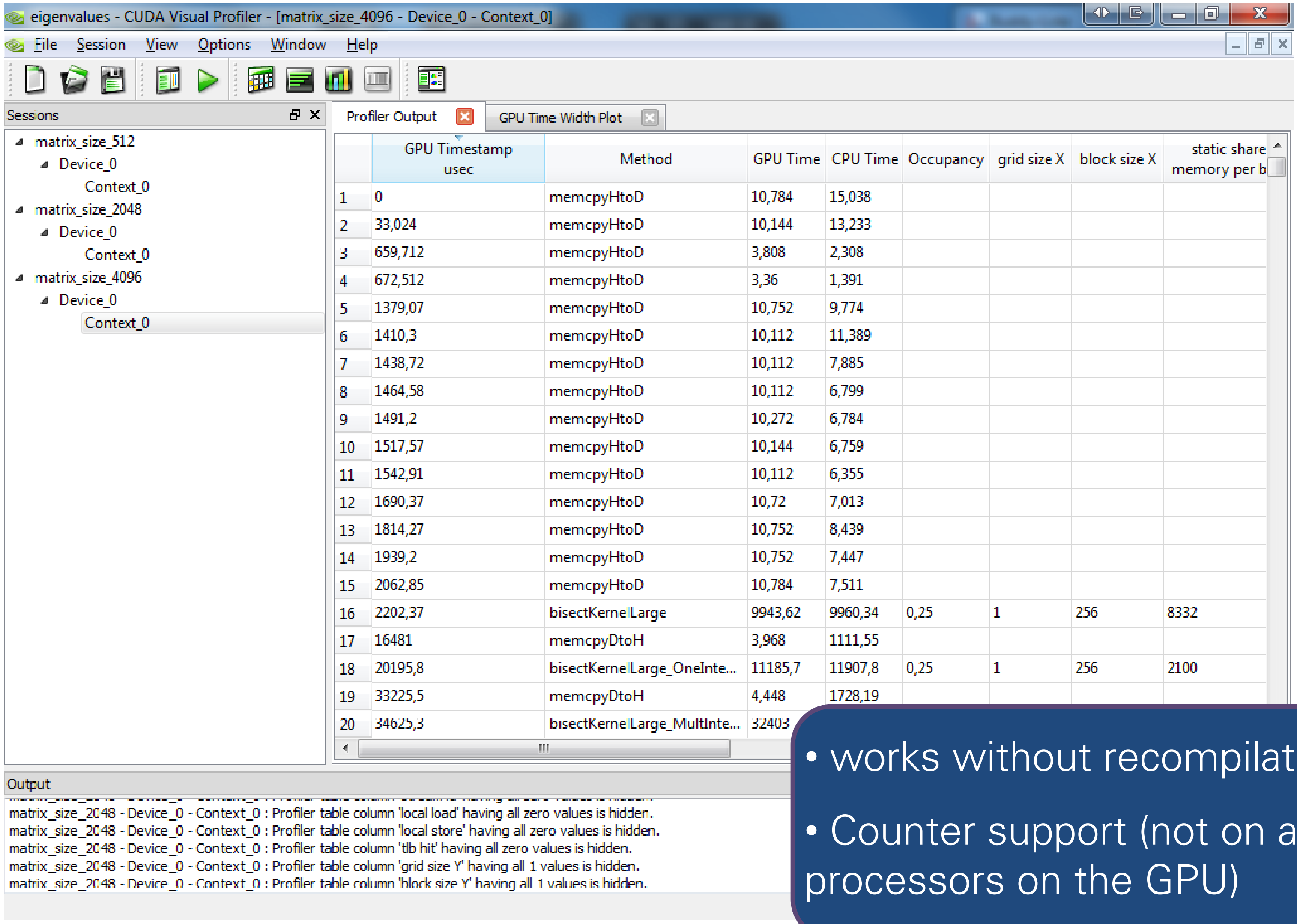
Performance monitoring hardware on the accelerator not/poorly exposed to user over the API

Using accelerators adds another level of complexity to (parallel) programming

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH
Center for Information Services & High Performance Computing

CUDA Visual Profiler (Eigenvalues; Profiler Output)



	GPU Timestamp usec	Method	GPU Time	CPU Time	Occupancy	grid size X	block size X	static share memory per b
1	0	memcpyHtoD	10,784	15,038				
2	33,024	memcpyHtoD	10,144	13,233				
3	659,712	memcpyHtoD	3,808	2,308				
4	672,512	memcpyHtoD	3,36	1,391				
5	1379,07	memcpyHtoD	10,752	9,774				
6	1410,3	memcpyHtoD	10,112	11,389				
7	1438,72	memcpyHtoD	10,112	7,885				
8	1464,58	memcpyHtoD	10,112	6,799				
9	1491,2	memcpyHtoD	10,272	6,784				
10	1517,57	memcpyHtoD	10,144	6,759				
11	1542,91	memcpyHtoD	10,112	6,355				
12	1690,37	memcpyHtoD	10,72	7,013				
13	1814,27	memcpyHtoD	10,752	8,439				
14	1939,2	memcpyHtoD	10,752	7,447				
15	2062,85	memcpyHtoD	10,784	7,511				
16	2202,37	bisectKernelLarge	9943,62	9960,34	0,25	1	256	8332
17	16481	memcpyDtoH	3,968	1111,55				
18	20195,8	bisectKernelLarge_OneInte...	11185,7	11907,8	0,25	1	256	2100
19	33225,5	memcpyDtoH	4,448	1728,19				
20	34625,3	bisectKernelLarge_MultInte...	32403					

```
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'local load' having all zero values is hidden.
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'local store' having all zero values is hidden.
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'tlb hit' having all zero values is hidden.
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'grid size Y' having all 1 values is hidden.
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'block size Y' having all 1 values is hidden.
```

- works without recompilation
- Counter support (not on all processors on the GPU)

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

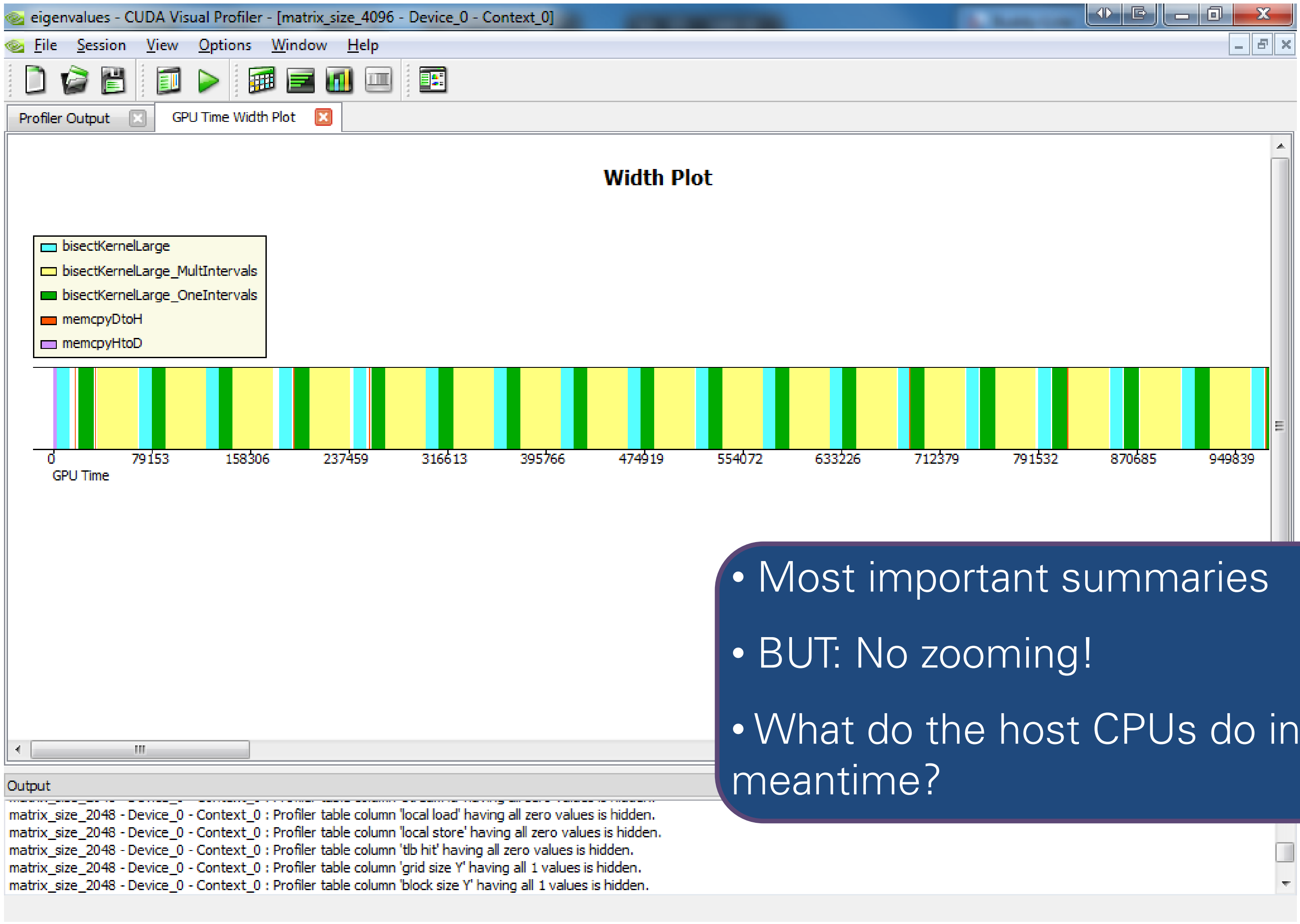
CUDA Profiler

- Profiling possible in every Cuda-Application without recompilation
  - Control via environment variables and config file
  - Output file with counters will be generated
  - Different counter values collectable BUT only 4 on one run
  - Multiple runs to collect all counters
- Counters
  - For single multiprocessor
    - Cumulative count for all thread blocks on multiprocessor 0
    - Examples: **branch**, **instructions**
  - For a Texture Processing Cluster (TPC); Examples:
    - **Gld uncoalesced**: Number of non-coalesced global memory loads
    - **Local load**: Number of loacal memory loads
    - **Tlb miss**: Number of instruction or constant memory cache misses

CUDA Visual Profiler

- GUI for profiling, based on Qt
  - Simplifies usage:
    - No extra config files
    - Multiple runs automatically
    - Charts for visual analysis
  - Features
    - GPU Time Summary Plot: find long running kernels
    - GPU Width Plot: simplified Timeline (GPU, memcpy only)
    - Comparison plots (summary for two sessions)
  - Advantages
    - Easy to use, works on all cuda-platforms (Windows, Linux, Mac)
  - Disadvantages
    - No zooming
    - Multiple runs
    - No support for CPU-Tracing

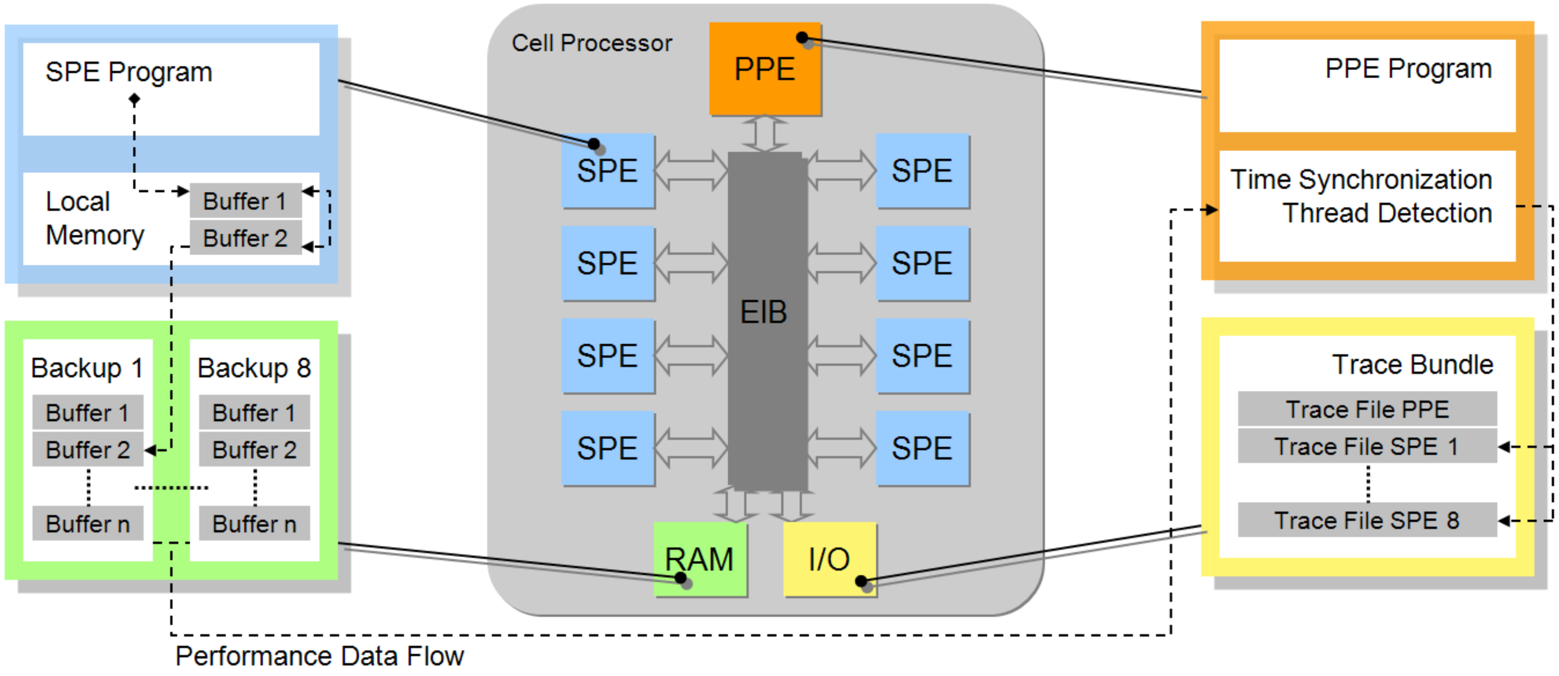
CUDA Visual Profiler (Eigenvalues; Width Plot)


eigenvalues - CUDA Visual Profiler - [matrix_size_4096 - Device_0 - Context_0]
File Session View Options Window Help
Profiler Output
GPU Time Width Plot
Width Plot
bisectKernelLarge
bisectKernelLarge_MultIntervals
bisectKernelLarge_OneIntervals
memcpyDtoH
memcpyHtoD
0
79153
158306
237459
316613
395766
474919
554072
633226
712379
791532
870685
949839
GPU Time
Output

```
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'local load' having all zero values is hidden.
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'local store' having all zero values is hidden.
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'tlb hit' having all zero values is hidden.
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'grid size Y' having all 1 values is hidden.
matrix_size_2048 - Device_0 - Context_0 : Profiler table column 'block size Y' having all 1 values is hidden.
```

- Most important summaries
- BUT: No zooming!
- What do the host CPUs do in the meantime?

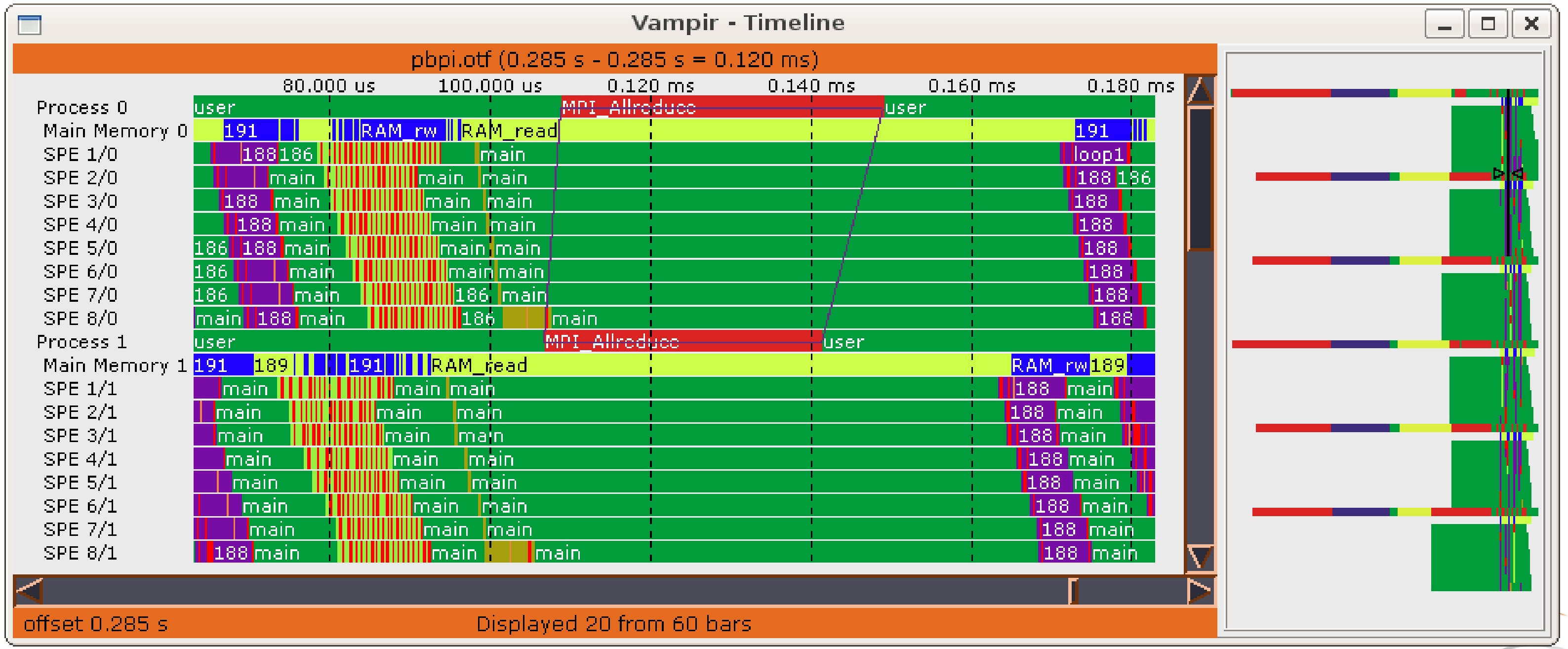
Special Solution: VampirTrace for CellBE



- Concept for Performance Tracing on Cell/B.E.
  - CellTrace, prototype integration in VampirTrace
  - Typical overhead: less than 5 percent
  - Fine grained event recording (resolution within 100s of nanoseconds)

CellBE Traces visualize in „Standard“ Vampir/VampirServer

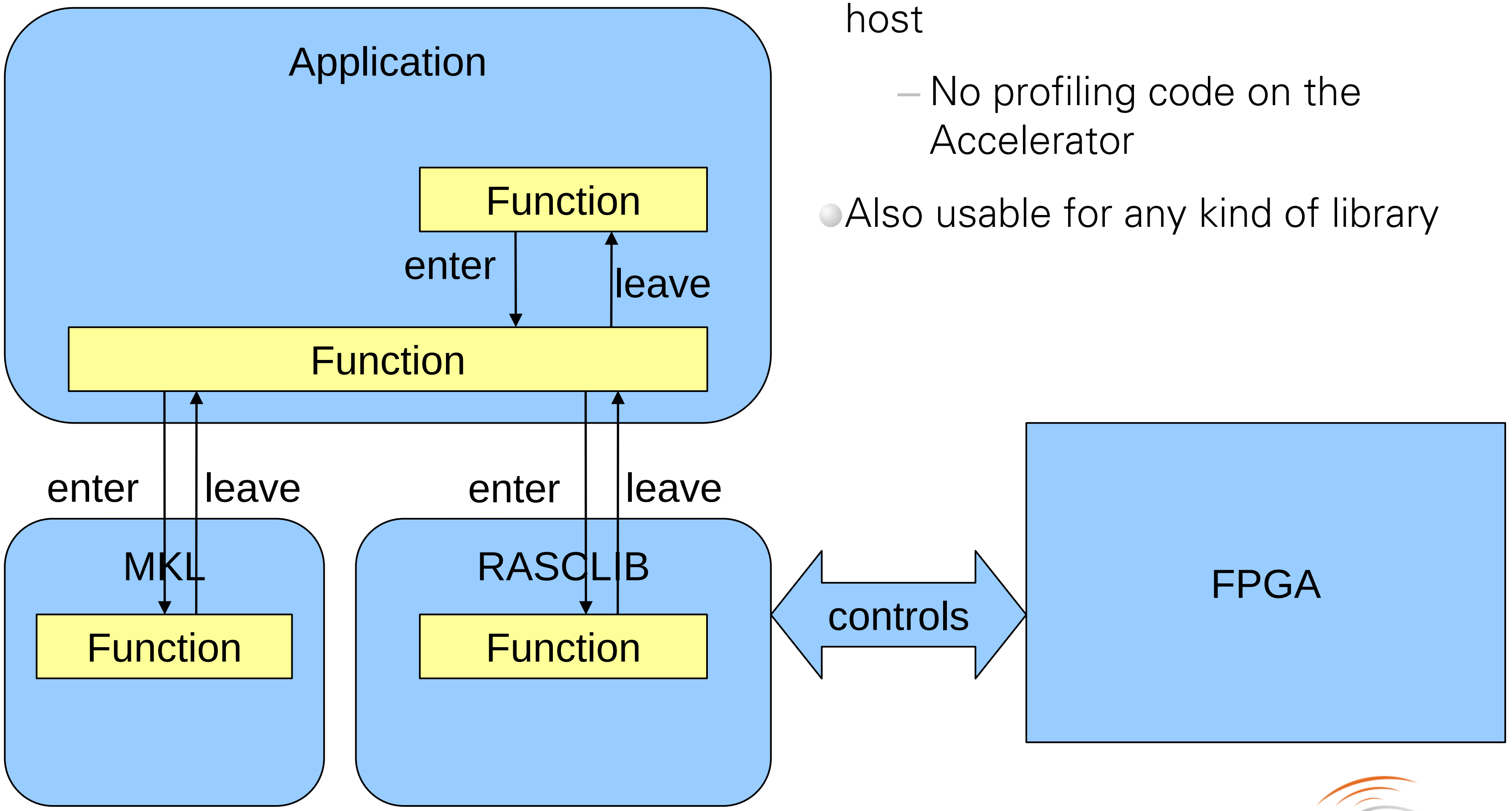
- Visualization of Traces with Vampir
  - Creates valuable insight into the runtime behavior of Cell applications
  - Intuitive performance visualization and verification
- Support for large applications with hybrid Cell/B.E. and MPI parallelism



VampirTrace API Tracing

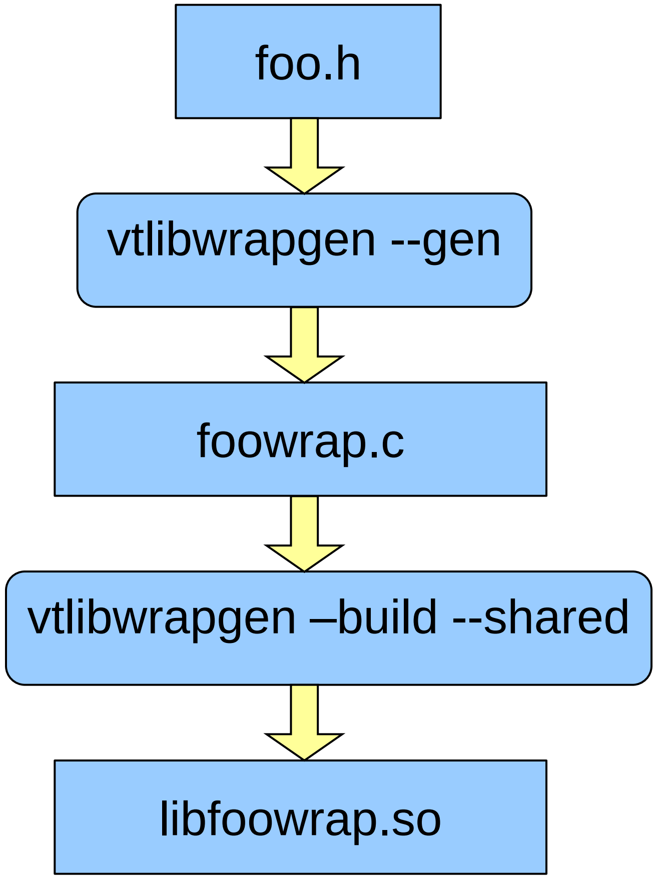
- Available since VampirTrace 5.8

Motivation



- Visualize Co-Processor usage by looking at library calls from the host
  - No profiling code on the Accelerator
- Also usable for any kind of library

Workflow



- Header-File of a(ny) library as source
- Generated wrapper functions use VampirTrace API
- Wrapper functions can be adapted (to include more information)
- Also works for static builds

LD_PRELOAD=$PWD/libfoowrap.so <executable>

Filter

- Some libraries are quite large (MKL 10: 7979 function declarations)
- Apply VampirTrace filters also to wrapper generation
- Usable for function names and file names in/of the header file
- Top down priorities

```
/usr/* -- 0
cblas_* -- -1
clapack_* -- -1
```

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

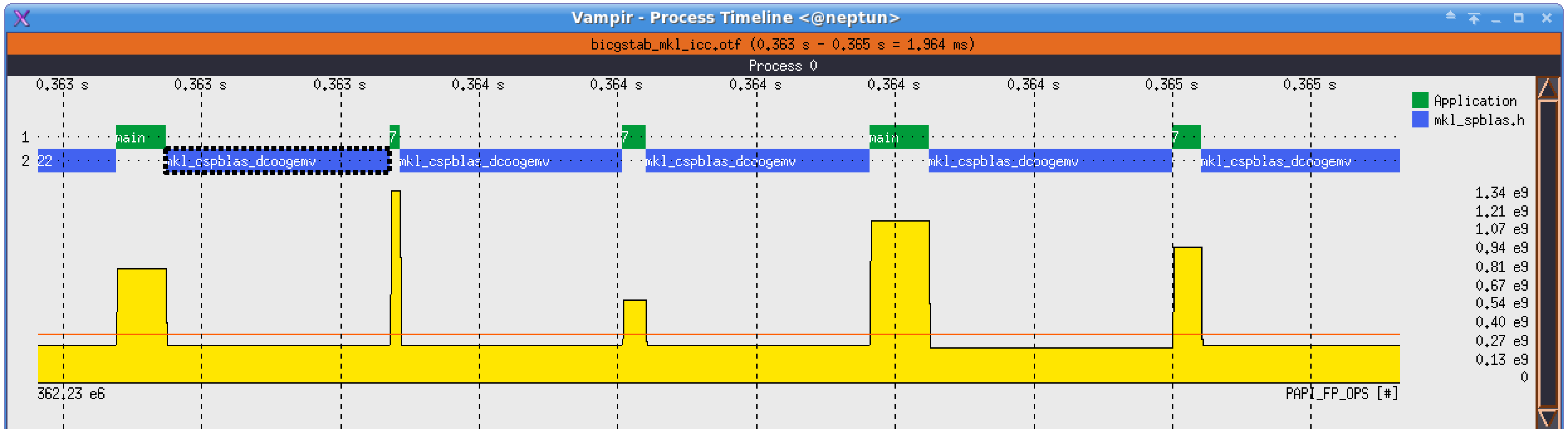
Example: Bicgstab

- Iterative equation solver
- MKL for matrix-vector-operations on sparse matricies
- Uses only one MKL header file: mkl_spblas.h

What you need to do:

- vtlibwrapgen --gen –o mklwrap.c -f filter.default $MKL_INC/mkl_spblas.h
- vtlibwrapgen –build –shared –o libmklwrap mklwrap.c
- LD_PRELOAD=$PWD/libmklwrap.so ./bicgstab

What you get:



VampirTrace CUDA Support

- Special Beta of VampirTrace 5.8
- Supports CUDA Versions 2.1-2.3

TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services & High Performance Computing

Beta Release (our Christmas present ☺)

Currently only CUDA Runtime API-Tracing

- Supports CUDA 2.1-2.3
- Runtime API hidden behind CUDA compiler (subject to change)

Use known metrics for parallelization

- Map GPUs to Threads
- Memory transfers = Thread communication

Statistics reusable

- Message statistics, summary charts
- But confusing when adding MPI and CPU threads (overlapping metrics)

Further plans

Counter support

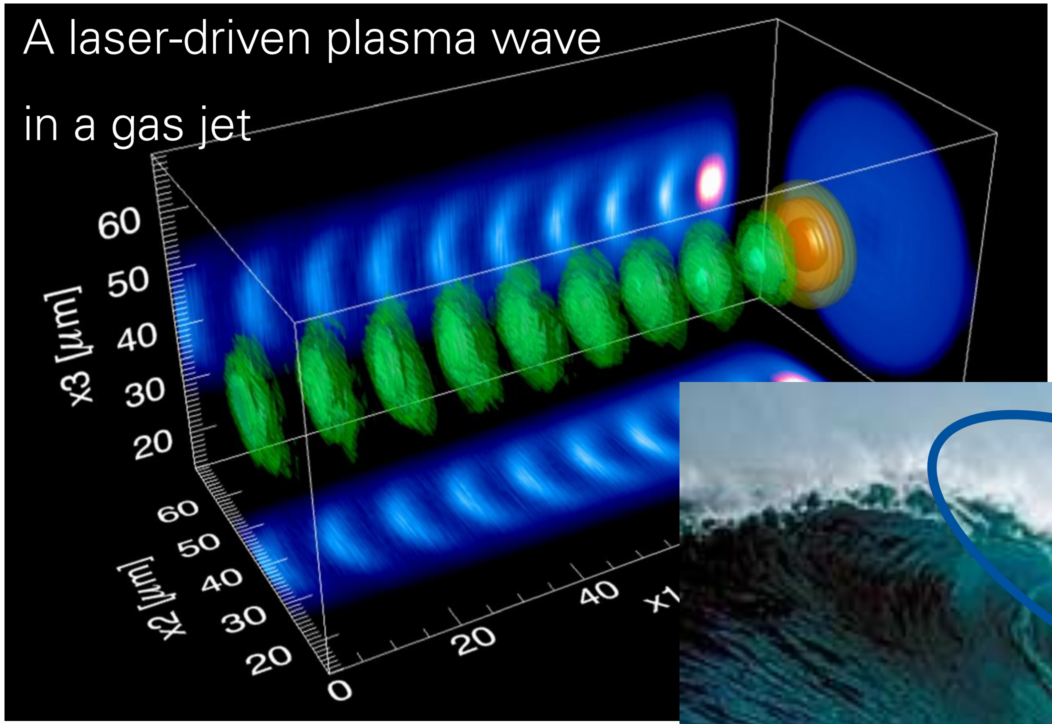
- Difficult, currently only possible by post-mortem OTF merging

Support multiple GPU Streams

Visualize more GPU activity

An example: PiConGPU

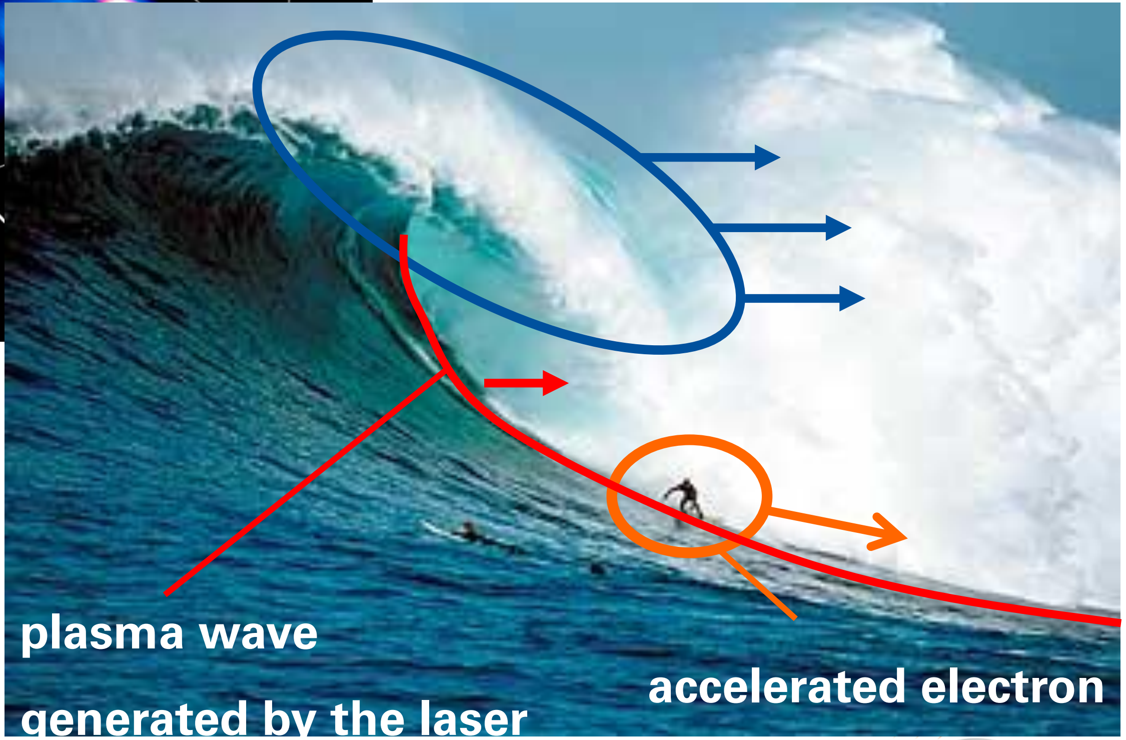
An Example: The Physics for Electron Acceleration



The Laser Pulse drives

wave-like Modulation

of the Electron Density



Electrons surf

on this Plasma Wave

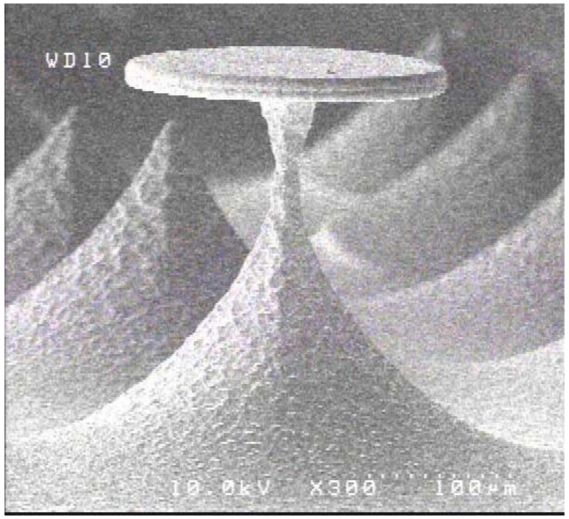
which trails the Laser Pulse

At the Speed of Light

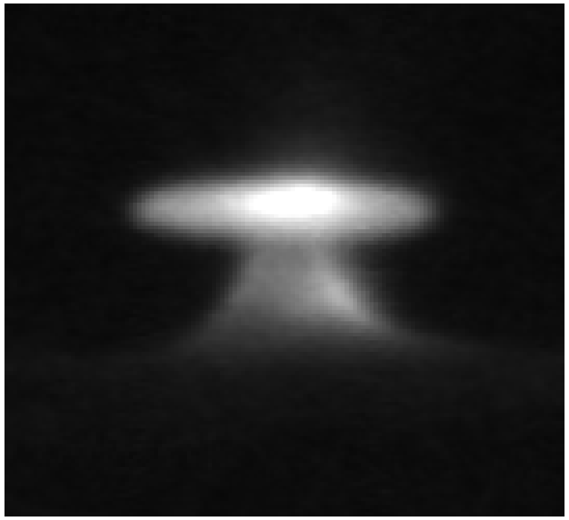
O. Jäkel, DKFZ Heidelberg

Simulation vs. Experiment (Pizza-Cone-Targets)

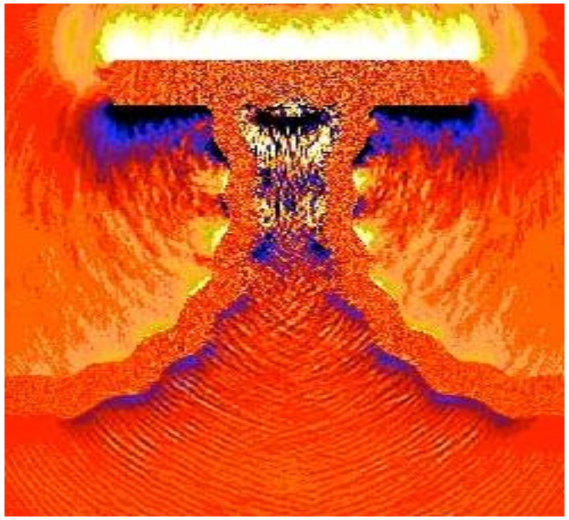
We know what we are doing…



Microscopic Image



X-ray Emission during Experiment



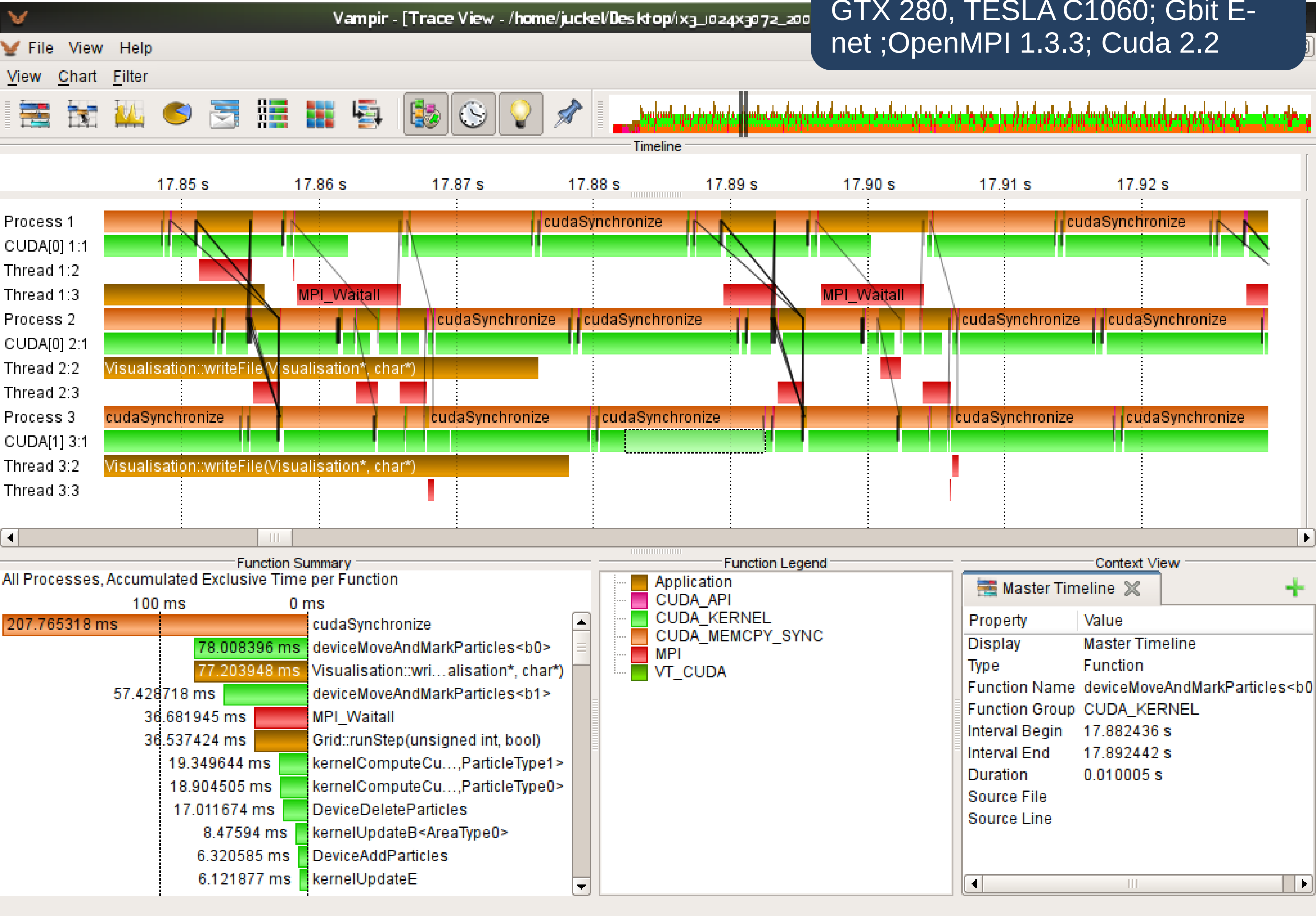
Simulation of Field Distribution

…but it is hard to experimentally probe the temporal evolution of the laser-target interaction on a time scale of femtoseconds

O. Jäkel, DKFZ Heidelberg

What does it looks like in Vampir?

Core i7 920 2.66 Ghz, 8GB RAM; GTX 280, TESLA C1060; Gbit E-net ;OpenMPI 1.3.3; Cuda 2.2



Function Summary

All Processes, Accumulated Exclusive Time per Function

100 ms … 0 ms	
207.765318 ms	cudaSynchronize
78.008396 ms	deviceMoveAndMarkParticles<b0>
77.203948 ms	Visualisation::wri…alisation*, char*)
57.428718 ms	deviceMoveAndMarkParticles<b1>
36.681945 ms	MPI_Waitall
36.537424 ms	Grid::runStep(unsigned int, bool)
19.349644 ms	kernelComputeCu…,ParticleType1>
18.904505 ms	kernelComputeCu…,ParticleType0>
17.011674 ms	DeviceDeleteParticles
8.47594 ms	kernelUpdateB<AreaType0>
6.320585 ms	DeviceAddParticles
6.121877 ms	kernelUpdateE

Function Legend

- 🟧 Application
- 🟪 CUDA_API
- 🟩 CUDA_KERNEL
- 🟧 CUDA_MEMCPY_SYNC
- 🟥 MPI
- 🟩 VT_CUDA

Context View

Master Timeline

Property	Value
Display	Master Timeline
Type	Function
Function Name	deviceMoveAndMarkParticles<b0
Function Group	CUDA_KERNEL
Interval Begin	17.882436 s
Interval End	17.892442 s
Duration	0.010005 s
Source File	
Source Line	

Summary/ Outlook

What has been accomplished?

Currently Solutions for three platforms in varying detail

- Cell -> extremely good (Standard Vampir, Special VampirTrace)
- GPU / FPGA -> only API tracing

Very low overhead

- Between 1% (GPU/FPGA) and 5% (Cell)

No need to learn new tools

- Interesting metrics are still the same (just a little more complex)

A couple of success stories with the new possibilities

- Understanding why FPGABLAST was so bad on our system
- A PiC implementation that the whole plasma physics community is extremely excited about
- A great tool to understand Cell applications

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

A Look into the Future

Support for OpenCL

Scalibility of the tool in general

- Expand to more processes for tracing and visualization

Requests from other groups

- Help on parallelizing their CUDA codes
- Performance studies on their hybrid codes

Acknowledgements

Daniel Hackenberg (ZIH)

- Cell Tracing

Rene Widera (ZIH)

- PiConGPU (Linux port and Control infrastructure)

Thomas Ilsche (ZIH)

- API-Tracing
- RASCLIB VampirTrace support

Wolfgang Hönig (ZIH)

- PiConGPU (MPI parallelisation)
- VampirTrace integration of CUDA

Heiko Burau (FZD)

- PiConGPU work (Physics and CUDA kernels)